Debreceni Egyetem
Informatika Kar
Informatikai rendszerek és hálózatok tanszék

Webes alkalmazásfejlesztés: könyvtár-információs rendszer

Témavezető:
Dr. Kuki Attila
egyetemi docens

Készítették:
Gönczi Krisztián
Bonta Mihály Zsolt
mérnök-informatikusok

Debrecen
2008

Köszönetnyilvánítás

Köszönetünket szeretnénk kifejezni az Epam Systems dolgozóinak, Bicskey Simonnak és Jávor Kornélnak, valamint témavezetőnknek Dr. Kuki Attilának, akik időt és energiát nem sajnálva voltak segítségünkre szakdolgozatunk megszületésében.

Abstract

The title of our degree work is: Developing a web based information system for libraries.

The main reasons why we decided to develop web application are:

- the wide range of possibilities
- advanced and most modern techniques can be used
- popularity and future of web

The project is based on ASP.NET technology, the code was written in C# 2008 programming language. We used Microsoft.NET 3.5 framework and Microsoft Visual Studio 2008.

Our goal was to get familiar with the main components and functions of ASP.NET, and to try to learn a lot of new features of C# language.

We have got acquainted with the using of master and content pages, ADO.NET, profiles, membership API, roles and security management, website deployment.

The software we have built has all the basic functions what a library needs for managing customers, books, loans and employees.

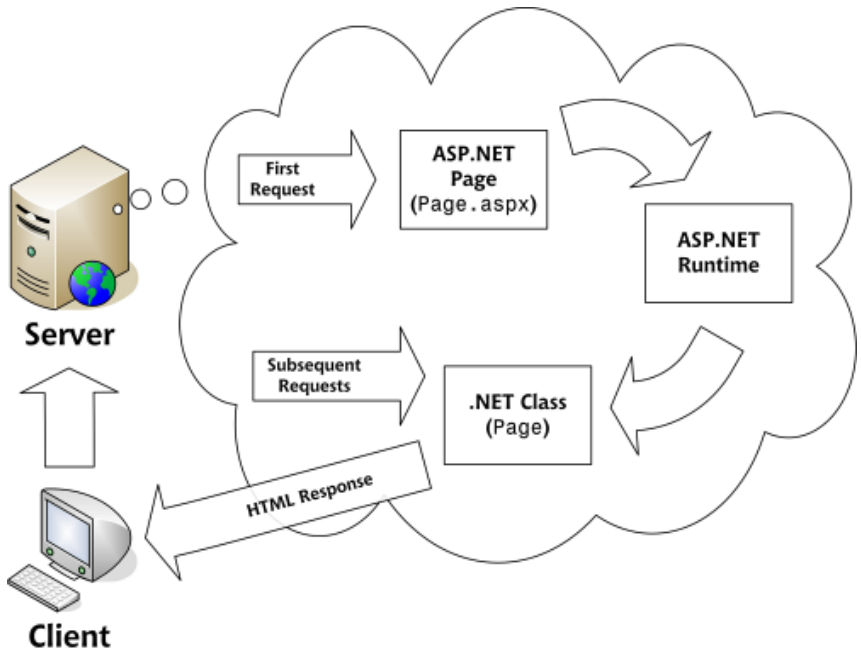
Tartalomjegyzék

1. Bevezetés

Napjainkban az internet használata és az ezzel összefüggő fejlődés elképesztő mértékű. A megnövekedett felhasználói igények minél teljesebb kielégítése mellett egyre nagyobb hangsúlyt fektetnek a fejlesztői oldal hatékonyabbá tételére. A webprogramozás manapság a szoftverfejlesztés legszerteágazóbb területe, köszönhetően a benne rejlő lehetőségek széles skálájának és a rugalmas programozhatóságnak. A programozók között talán a webfejlesztő az, akinek a legsokoldalúbbnak kell lennie, mivel egy adott probléma megoldására számos eszköz, nyelv, technológia áll rendelkezésére. Szinte lehetetlen ezek mindegyikét megismerni és jól alkalmazni, így első lépésként nagyon fontos, hogy eszköztárunkból jól válasszuk ki azokat az eszközöket, amelyekkel a leghatékonyabban tudjuk megoldani az előttünk álló feladatot. Az interneten a távoli elérések és a felhasználók nagy száma miatt nagyon sok tényező befolyásolja a sebességet, a kompatibilitást, a megbízhatóságot, ezért itt különösen fontos az optimalizálás, az akadálymentesítés. Mindezek miatt a második lépés az, hogy a megfelelően választott eszközöket hatékonyan alkalmazzuk, a cél a felmerülő igények minél tökéletesebb kielégítése kell, hogy legyen.

Manapság a webes nyelvek és technológiák fénykorukat élik. Folyamatosan új, egyre hatékonyabb eszközök jelennek meg, a már meglévők pedig állandó fejlesztéseken, bővítéseken mennek keresztül. A dinamikus, interaktív webszolgáltatások megjelenésével előtérbe került a szerver oldali programozás, így a legnagyobb előretörést ezen a téren tapasztalhatjuk. Míg például a Perl folyamatosan háttérbe szorul, továbbra is hódít a PHP, emellett pedig a legnagyobb népszerűségnek a keretrendszerek örvendenek, amik segítségével egyszerűbben, gyorsabban és kényelmesebben lehet megfelelni az előbbiekben említett, webes alkalmazásokkal szemben támasztott elvárásoknak. 2004-ben megszületett a Ruby on Rails, amely jó példa arra, hogyan lehet egy nem kifejezetten webes nyelvet - ez esetben a Ruby-t - alkalmassá tenni arra, hogy segítségével igen hatékonyan írhassunk szerver oldali szkripteket. A Rails sikerén felbuzdulva egyre többen kezdtek hasonló projektekbe, így a Rails mintájára fejlesztik többek között a CakePHP-t, vagy a Python nyelvhez a Django-t. Igazi jelentőségük abban mutatkozik meg, hogy segítségükkel sok programozó előtt nyílik meg a lehetőség a webfejlesztésre anélkül, hogy az általuk jól ismert és kedvelt programozási nyelveket nélkülözniük kellene.

Hasonló céllal jött létre a Microsoft.NET keretrendszer is, amelynek minden előnyét kihasználva alakult ki az ASP.NET (Active Server Pages) webfejlesztésre szánt technológia. A .NET-tel a Microsoft egy olyan integrált csomagot alkotott, amely kombinálja a web építőelemeinek számító HTTP-t és a leíró nyelveket az objektum-orientált módszertannal. Az ASP.NET segítségével létrehozott webes alkalmazások esetén az egyszerű weblapoknál megszokott módon érvényesülnek a hagyományos koncepciók, tehát lehetőség van például HTML vagy JavaScript használatára is. Lássuk most az ASP. NET néhány olyan tulajdonságát, amely megkülönbözteti a hagyományos technológiáktól! Objektum-orientált programozási modell áll mögötte, amely egy eseményvezérelt, vezérlőkre épülő szerkezetet foglal magába, elősegítve ezzel a programkódok egymásba ágyazásának és újrafelhasználásának lehetőségét. Lehetőséget nyújt arra, hogy a programozó bármely a .NET által támogatott programnyelven kódoljon. Ezek a nyelvek alapvetően a C#, Visual Basic és J#, de lehetőség van akár Ruby vagy Python nyelven is programozni az IronPython és IronRuby implementációknak köszönhetően. Az ASP.NET kódot fordítás után a CLR (Common Language Runtime) futtatja, egy olyan futtatórendszer, amely biztosítja az automatikus memóriakezelést és szemétgyűjtést, ügyel a típusbiztonságra, kiterjedt hibakezelést nyújt és lehetőséget biztosít a többszálú működésre.



1. ábra

Egy ASP.NET weboldal életciklusa

Az általunk készített alkalmazás fejlesztéséhez is az ASP.NET technológiát használtuk, amelyre az előbb említett megfontolásokból esett választásunk. A kódolás HTML, CSS és C# programozási nyelven történt, utóbbit korábbi tanulmányaink során ismertük meg és a Microsoft Visual Studio-val kiegészítve nagyon hamar kedvenc fejlesztőeszközünkké vált. Úgy gondoltuk, hogy egy könyvtári rendszer az, amely kellőképpen egyszerű és összetett funkciókat is tartalmaz ahhoz, hogy az ASP.NET-tel való ismerkedésünk során, fokozatosan létre tudjuk hozni azokat, az egyszerűbbektől a bonyolultabb megoldások felé haladva. A munkát a tervezési fázissal kezdtük. Első lépésként megtárgyaltuk azokat a funkciókat, amelyekkel a szoftver rendelkezni fog. Ezt követően meghatároztuk a rendszert használók körét, aminek eredményeképpen négy szerepkört alakítottunk ki. Következett az adatbázis tervezése. Tudtuk, hogy ez létfontosságú művelet a későbbiek szempontjából, mivel egy ilyen rendszernek amellett, hogy nagyméretű adatbázissal dolgozik, meg kell birkóznia a jelentős mértékű adatforgalommal is. Ezzel párhuzamosan alakítottuk ki a webes felületet, amelynél a lehető legegyszerűbb megjelenésre törekedtünk, olyan szempontokat szem előtt tartva, mint az átláthatóság és a könnyű kezelhetőség. Csak mindezek után kezdtük meg a tényleges funkciók megvalósítását. Ennek során próbáltuk modulokra osztani a rendszert és egy-egy modul elkészülése után megvizsgáltuk, hogy az logikailag kapcsolatban van-e egy másik, már meglévő egységgel. Amennyiben igen, úgy kiépítésre kerültek a tényleges kapcsolatok, összeköttetések. Végezetül beállítottuk a hozzáféréseket és a szükséges biztonsági eszközöket.

Dolgozatunkban egy rövid rendszerismertetőt követően részletesen beszámolunk a komponensekről, az általunk alkalmazott eszközökről és technikákról, a fejlesztés során felmerült problémákról, buktatókról. Minden esetben kitérünk az adott fejezetre vonatkozó ASP.NET eszközökre és azok elméleti működésére, amelyekre gyakorlati példaként az általunk készített szoftver oda tartozó része kerül bemutatásra. Írásunk így az ASP.NET működésének és lehetőségeinek áttekintéseként is szolgál.

2. A rendszer rövid ismertetése

Szoftverünk egy hagyományos felépítésű webes alkalmazás, amely weblapok sokaságát foglalja magába. Ezek a weblapok csoportosítva vannak, a csoportokat a szerepkörök és az ezekhez kapcsolódó jogosultságok határozzák meg. Az egyes oldalakon található vezérlők kapcsolatban állnak a rendszer mögött álló adatbázissal, így biztosítva az adatok kezelését. A rendszer egy átlagos könyvtár működéséhez szükséges adatkezelési, adattárolási szolgáltatásokat, illetve információ nyújtásához szükséges alapvető eszközöket tartalmaz. Az alkalmazás a megfelelő beállítások után az interneten keresztül érhető el, de kisebb módosításokkal belső hálózaton való működésre is alkalmas. A megvalósítás során két fajta hozzáférési módot és négy szerepkört különítettünk el.

Hozzáférési módok:

- publikus: a legtágabb hozzáférés, amely lehetőséget biztosít arra, hogy bizonyos oldalakat bárhonnan elérjünk egy böngészőn keresztül. Ily módon lehetőség van az aktuális hírek olvasására, a könyvek adatbázisában való keresésre és böngészésre, beiratkozott ügyfelek részére a belső felületre való belépésre, valamint a könyvtárral kapcsolatos elérhetőségek, információk, szabályzatok megtekintésére.
- nem publikus: minden egyéb, belső, autentikációt igénylő oldalt célzó hozzáférés tartozik ide. Az ilyen típusú hozzáférések szerepkörök szerint vannak tovább osztva.

Szerepkörök:

- privát (Private): ebbe a szerepkörbe tartoznak a könyvtárba beiratkozott ügyfelek. A beiratkozás során automatikusan regisztrálódnak a rendszerben és elérhetik saját belső oldalukat, ahol a kölcsönzéseiket, üzeneteiket és felhasználói profiljukat találják.
- inaktív (Inactive): ide tartoznak azok az ügyfelek, akiknek beiratkozási idejük lejárt, így nem érhetik el a belső oldalakat, viszont személyesen meghosszabbíthatják regisztrációjukat, ami után ismét a privát szerepkörbe kerülnek.

- alkalmazott (Employee): a könyvtárban dolgozók szerepköre. Elérhető minden oldal, ami privát szerepkörrel elérhető, ezen kívül egyéb olyan adminisztrációs oldalak, mint az ügyfelek regisztrálása, hírek és üzenetek kezelése, könyvek adatbázisának kezelése. Ezeken kívül az alkalmazotti szerepkörben lévők jogosultak az adatok módosítására is.
- rendszergazda (Admin): ide tartoznak a rendszert kezelő, felügyelő személyek. Jogosultságaik a rendszer működése során felmerülő hibák kezelésére, a teljes ügyfél- és dolgozó-adatbázis módosítására, a hírek menedzselésére, üzenetek küldésére és fogadására, valamint statisztikák begyűjtésére terjednek ki.

Az általunk használt adatbázis kezelő a Microsoft SQL Server 2005 Express változata, amely lehetővé tette számunka, hogy a teljes adatbázisunkat egyetlen mdf (Master Database File) kiterjesztésű fájlban tároljuk. Ez nagyban megkönnyítette a munkánkat, mivel a táblák és a benne lévő adatok szinkronizálását egyszerűen oldhattuk meg egymás között. Nagy segítségünkre volt még a verziókövetés, amelyhez Subversion-t használtunk, az online repository-t pedig az Assembla szolgáltatta. A szoftver az Internet Explorer 7 és a Firefox 3 típusú böngészőkre lett optimalizálva.

3. Adatkezelés

3.1 Adatbázis

Egy program fejlesztése során az egyik legnehezebb feladat a megfelelő adatbázis létrehozása. Nemcsak hatékonyságbeli problémák adódhatnak a rossz tervezésből, hanem nagy lehet a kockázat az adatok biztonsága, a teljes rendszer működőképessége terén is.

Az ügyfelek és dolgozók nyilvántartására, adataik kezelésére a beépített Membership, illetve Profile rendszert használtuk, az ezekhez tartozó adattáblákkal. Mivel ezekről később, egy külön fejezetben lesz szó, most lássuk az általunk létrehozott táblákat!

- Message: felhasználók küldött üzenetek táblája

  Attribútumai:

  - Sender: a feladó azonosítója

  - Recipient: címzett e-mail címe

  - Date: üzenet kézbesítésének időpontja

  - Subject: üzenet tárgya

  - Content: üzenet tartalma

- News: az oldalon megjelenő hírek táblája

  Attribútumai:

  - Date: hír létrejöttének dátuma

  - Title: hír címe

  - Valid: hír érvényessége

  - Text: hír tartalma

  - Sender: a hír feladója

- Books: a könyvtárban található könyvek adatainak tárolására szolgál.

  Attribútumai:

  - ID: a könyv egyedi, könyvtári sorszáma, egyben a tábla elsődleges kulcsa

  - ISBN: a könyv egyedi, nemzetközi azonosítója

  - Author: a könyv szerzője

  - Title: a könyv címe

  - Publisher: a könyv kiadója

  - Year: a könyv kiadásának éve

  - State: a könyv állapota: 0-elérhető, 1-kölcsönözve, 2-lefoglalva

  - Available: a könyv elérhetősége

- Loan: az aktuális kölcsönzések nyilvántartására szolgáló tábla.

  Attribútumai:

  - LoanID: az aktuális kölcsönzés azonosítója – Elsődleges kulcs

  - BookID: a kikölcsönzött könyv azonosítója – Külső kulcs

  - UserID: felhasználó azonosítója – Külső kulcs

  - From: kölcsönzés kezdete

  - To: a kölcsönzés vége

  - BackDate: kölcsönzés tényleges vége

- Reserve: az aktuális foglalások nyilvántartására szolgáló tábla

  Attribútumai:

  - ReserveID: a foglalás azonosítója – Elsődleges kulcs

  - BookID: a lefoglalt könyv azonosítója – Külső kulcs

  - UserID: a felhasználó azonosítója – Külső kulcs

  - From: a lefoglalás ideje

  - To: a lefoglalás vége

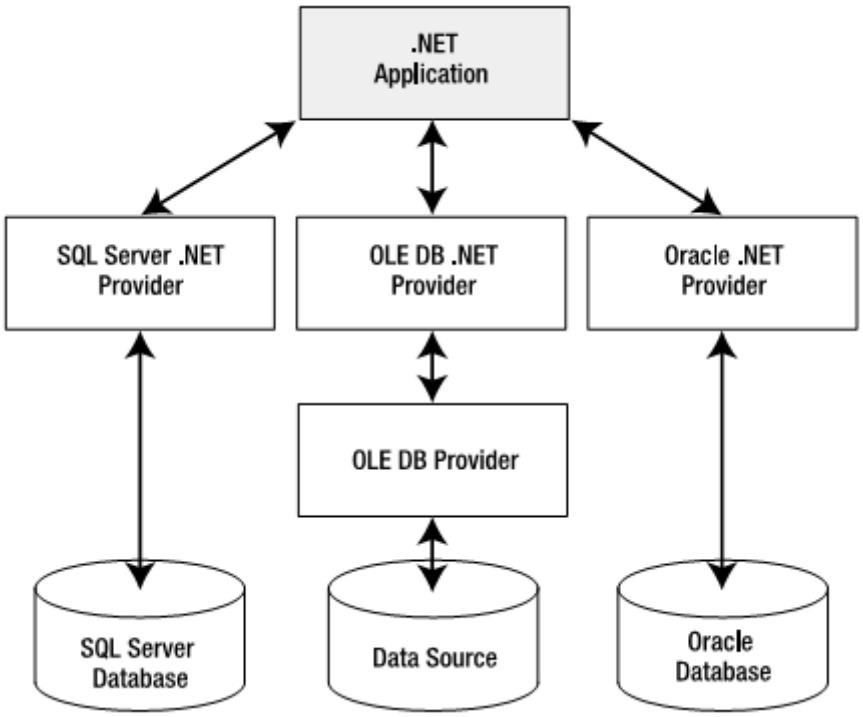
Az adatbázis tervezésénél egyszerű modellre törekedtünk, ahol a redundancia teljes mértékben való megszüntetése nem volt célunk; de a különféle anomáliákra szerettünk volna megoldást találni. Egy összetett könyvtári-információs rendszerhez képest, mi csak a könyvtári modell legfőbb jellemzőit emeltünk ki. Így kimaradt például az ETO (Egyetemes Tizedes Osztály) jelölés, amely a könyvek rendszerezésére szolgál, továbbá az ISSN és a raktári szám. A tárhely igényeket háttérbe szorítva, célunk minél gyorsabb műveletek elvégzése volt, mivel a legtöbb oldalon a felhasználók valamilyen adatbázis tranzakciót hajtanak végre. Erre fókuszálva, a táblák egyes attribútumai indexelve vannak, amelyek ugyan gyorsítják a tranzakciókat, de a frissítés az indexelés időigényével megnő. A műveleteket ellenőrzött körülmények között, vezérlők és validator–ök segítségével mennek végbe, anélkül, hogy egyetlen DML utasítást is írnánk.

3.2 Adatok elérése és kötése

A számítógépes alkalmazások legnagyobb része - legyen az kliens vagy webes alkalmazás – adatvezérelt. Az adatok felhasználásának módja az utóbbi időben jelentősen megváltozott. A kisebb lokális adatbázisok helyét egyre inkább az elosztott rendszerekre

jellemző központosított adatbázisok vették át. Ennek megfelelően szükség volt az adatelérési technológiák átalakítására is.

A .NET keretrendszernek saját adatelérési technológiája van, az ADO.NET. Ez olyan osztályokat tartalmaz, amelyek segítségével lehetőség nyílik adatforrásokhoz - legtöbb esetben relációs adatbázisokhoz - való kapcsolódásra, adatbázis parancsok futtatására és a már kinyert adatok adatbázis-kapcsolat nélküli kezelésére. Az adatkezelési szolgáltatások lényegi működése az ASP.NET kezdeti verziójától kezdve nem módosult, a fő koncepció mindig is az volt, hogy a lehető legkevesebb kód írásával érjük el célunkat.



2. ábra

Az ADO.NET architektúra

Az ADO.NET eszközeit alapvetően két típusba sorolhatjuk: kapcsolat-alapú, illetve tartalom-alapú objektumok. Előbbi csoportba tartozik például a Connection, a Command, a DataReader vagy a DataAdapter, az utóbbinak része a DataSet vagy a DataRelation. Fontos különbség közöttük, hogy míg a kapcsolat-alapú objektumok adatforrás specifikusak, tehát alkalmazásukat befolyásolja az, hogy honnan szeretnénk adatot kinyerni, addig a tartalom-alapúak adatforrástól függetlenek.

Az adatbázist elérhetjük anélkül is, hogy közvetlenül használnánk az ADO.NET osztályait. Erre többféle lehetőségünk is van:

- Az SqlDataSource vezérlő: lehetőséget nyújt deklaratív lekérdezések definiálására. Egyéb, például az adatok megjelenítését szolgáló vezérlőkhöz kapcsolható, így alkalmunk nyílik az adatok szerkesztésére, módosítására anélkül, hogy ehhez kódot kellene írnunk.

- LINQ to SQL: a 3.5-ös .NET Frameworkben megjelenő LINQ (Language Integrated Query) használatával programkódban, C# nyelven írhatunk adatbázis lekérdezést.

- Profilok: ez a mód meglehetősen korlátozott, olyan szempontból, hogy a szükséges táblákat, a hozzájuk tartozó tárolt utasításokat és az adatmanipulációs műveleteket a rendszer automatikusan kezeli. Kifejezetten felhasználói adatok tárolására készült.

Munkánk során kipróbáltuk és megtanultuk a közvetlen kóddal történő adatkezelést, de túlnyomórészt az SqlDataSource vezérlőt használtuk, míg az ügyfelek adatainak kezelésére a profilokat vettük igénybe.

A fejlesztés alatt az első adatkezeléssel járó feladatunk a hírek kezdőlapra való listázása volt. Itt az adatelérés lekódolását választottuk. A feladat az volt, hogy a News táblában található hírek közül felsorolás formájában a kezdőlapra kerüljenek azok a hírek, amelyek érvényessége még nem járt le, vagyis a ’Valid’ nevű attribútumuk értéke nagyobb, mint az aktuális dátum. Az adatbázissal való kapcsolat kiépítésének első lépése a connection string meghatározása. Ez a string név-érték párok pontosvesszővel elválasztott sorozatából áll, segítségével megadhatók és rögzíthetők azok az adatbázis-specifikus beállítások, amelyekre az adatbázissal való kapcsolat kiépítéséhez szükségünk lesz.

Mivel ezek a beállítások adatbázisonként nagyban eltérhetnek, jó szolgálatot tehet a http://www.connectionstrings.com/ weboldal, ahol az összes elterjedt adatbázis-kezelő rendszerhez szükséges string megtalálható. A connection string a web.config fájlban kerül elhelyezésre. A web.config az ASP.NET alkalmazások fő konfigurációs fájlja. Típusát tekintve egy XML fájl, ami az alkalmazás beállításait tartalmazza. Ide kerülnek többek között a biztonsági beállítások, a session állapot konfigurációja, a programnyelvi és a fordításra

vonatkozó beállítások is. A fájlt az alkalmazás gyökérkönyvtárában kell tárolni, de bármely alkönyvtárnak lehet saját web.config-ja, amely az adott alkönyvtárban lévő weblapokra lesz érvényes és azokra vonatkozóan felülbírálja a fő konfigurációs fájl beállításait. Esetünkben a connection string a következőképpen néz ki:

```
<connectionStrings>

<add name="Library" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Library.mdf;Integrated
Security=True;User Instance=True" />

</connectionStrings>
```

Tartalmazza a szerver nevét, ahol az adatbázis található, az adatbázis nevét és elérési útvonalát, illetve az autentikációhoz szükséges beállításokat. A következő lépés a kapcsolat felépítése. Ehhez a System.Data.SqlClient névtér használata szükséges. Példányosítanunk kell az SqlConnection osztályt, amely konstruktora paraméterként várja a connection stringet. A konfigurációs fájlban megadott stringet az alábbi módon érhetjük el kódból:

```
WebConfigurationManager.ConnectionStrings["Library"].ConnectionString;
```

Ezzel definiáltuk a kapcsolatot, most szükségünk lesz a kiadandó SQL utasításra és egy SqlDataAdapter-re, ami ahhoz szükséges, hogy az adatbázisból kinyert adatokat egy DataSet-be töltsük. A DataSet egy olyan eszköz, ami táblázatos felépítésű és segítségével lehetőség van az adatok kezelésére azután is, miután már nem kapcsolódunk az adatbázishoz. Az SQL parancs létrehozásához az SqlCommand osztályt használjuk, aminek példányosításakor szövegesen adhatjuk meg a végrehajtandó utasítást, illetve szükség van a már előzőleg létrehozott, kapcsolatot reprezentáló SqlConnection példányra. Az SQL utasítások szöveges megadásán kívül lehetőség van tárolt utasítások létrehozására is. Ezeket mi magunk írhatjuk meg és az adatbázisunkban egy külön ezek számára fenntartott helyen kerülnek tárolásra (Stored Procedures).

Az ilyen fajta utasítások tetszőleges összetettségűek lehetnek, egyszerre akár több lekérdezést, módosítást is tartalmazhatnak. Előnyük többek között az, hogy karbantartásuk egyszerű, mivel ha módosítanunk kell, elég azt egy helyen megtenni. Ezenkívül megfelelnek a biztonsági elvárásoknak is és segítségükkel a rendszer teljesítménye is javítható. Kódból való elérésükhöz az SqlCommand példányosításakor az utasítás szövege helyett a tárolt utasítás

nevét kell megadnunk, illetve az így létrejött példány CommandType property-jének CommandType.StoredProcedure értéket kell adnunk.

Létrehozunk egy üres dataset-et, majd egy sqldataadaptert, amelynek meg kell adnunk a parancs és a kapcsolat objektumot, majd ezután már csak az adapter Fill metódusát hívjuk, ami paraméterként a dataset-et várja. Ezzel megtörtént a dataset feltöltése a lekérés eredményéül szolgáló adatokkal. Végezetül egy ListView vezérlő kerül elhelyezésre azon az oldalon, ahol a hírek lesznek láthatóak, majd ennek a vezérlőnek megadjuk, hogy az általa megjelenített adatok forrása az előbb feltöltött dataset legyen és a DataBind hívással megtörténik az adatok vezérlőhöz való kötése:

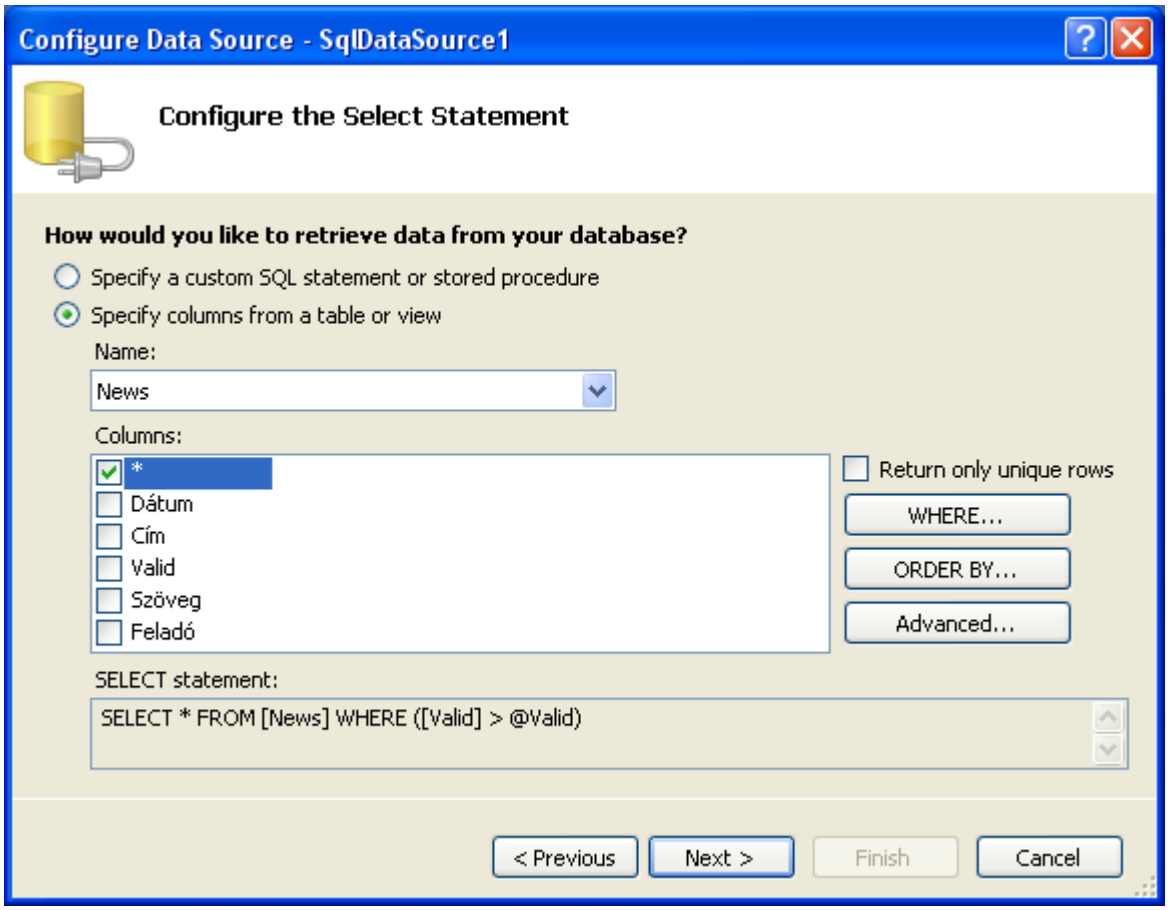
```
SqlConnection connection = new SqlConnection(connectionString);
DataSet dS = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter(new SqlCommand("SELECT * FROM
News WHERE Valid > GETDATE()", connection));

adapter.Fill(dS);

ListView1.DataSource = dS;
ListView1.DataBind();
```

Bár az előző megoldás egyszerű, komoly adatbázis műveletek esetén nagyon kódolásigényes lehet, megnövelve ezzel a hibalehetőségek számát és a fejlesztésre fordított időt. Az alapvető kapcsolódások, lekérések, módosítások kezelésére a szakirodalom is a datasource vezérlők használatát javasolja. Ezek közül a legtöbb esetben az SqlDataSource-t használtuk. Ez a vezérlő egy adatbázis kapcsolatot reprezentál. Használatához először be kell állítanunk a ProviderName tulajdonság értékét, amely adatbázis specifikus és a mi esetünkben SQL Server-ről lévén szó ez a System.Data.SqlClient lesz. Szükség van még az előzőekben tárgyalt connection stringre, és ezzel a kapcsolat beállításra került. Ezután beállíthatjuk a vezérlő Select, Insert és Delete utasításait, amiket majd az adatbázison hajt végre.

Mindezt grafikus felületen tehetjük meg, kód írása nélkül. Az alábbi ábrán az előbb kódban megadott kapcsolódási és lekérdezési procedúra látható grafikus felületen beállítva.



3. ábra

Select utasítás beállítása SqlDataSource-on a hírek lekérdezéséhez.

Az így konfigurált datasource-t bármilyen adatmegjelenítésre szánt vezérlőhöz kapcsolhatjuk oly módon, hogy az adott vezérlő DataSourceID tulajdonságának a datasource azonosítóját adjuk értékül.

Néhány mondat az általunk használt megjelenítő vezérlőkről:

- GridView: a legtöbb funkcióval rendelkező, táblázat alapú vezérlő. Az adatbázis rekordjait a tábla soraiban jeleníti meg. A sor oszlopai a megjeleníteni kívánt attribútumok lesznek. Az oszlopoknak nevet adhatunk és kiválaszthatjuk azt is mely attribútumokat szeretnénk megjeleníteni.

Támogatja a lapozást, az adatok rendezését valamint az adatok helyben történő módosítását. Lehetőség van más vezérlőket integrálni a gridview mezőibe.

- ListView: az ASP.NET 3.5-ös változatában jelent meg. Táblázat nélküli megjelenítést tesz lehetővé, nagy mértékben testre szabható, támogatja sablonok létrehozását és használatát.
- DetailsView: egyetlen rekord részletes, táblázatos formában való megjelenítésére szolgál. Támogatja a benne szereplő adatok módosítását és a sablonok használatát.

4. Megjelenés

4.1 XHTML és CSS

Szoftverünk vázát XHTML és CSS kód adja. A kinézet megtervezésekor igyekeztünk a lehető legszigorúbb szabványoknak megfelelni, figyelembe venni az adott leíró nyelvre vonatkozó kódolási konvenciókat. A szabványok betartása nem minden esetben sikerült, köszönhetően a böngészők közötti eltéréseknek. Nem ütköztünk sok problémába, mivel egyszerű az alkalmazásunk felépítése, ezért nem volt szükség komolyabb kód írására, viszont szomorúan tapasztaltuk és több forrás is megerősített minket abban, hogy a böngészők mennyire figyelmen kívül hagyják az egyes szabványokat saját érdekeik miatt, ezzel is nehezítve az optimalizálást és az akadálymentesítést. Az XHTML kódunk megfelel az 1.0 Transitional, míg a CSS kód a 2.1-es ajánlásoknak. Mivel az alkalmazásunkat alkotó weboldalak alapvető megjelenése csak minimálisan tér el egymástól, elegendő volt egyetlen CSS fájl használata.

4.2 Master és Content oldalak

Egy webes alkalmazás létrehozásakor nem elég a benne lévő weboldalak megjelenését egyenként megtervezni és kialakítani, hanem szükség van egy olyan eszközre, amely ezeket egyetlen egységbe fogja össze. Egy ilyen eszközre példa a master page, amely segítségével újrahasznosítható megjelenési minták hozhatók létre. A master page kialakításával megalkothatjuk az alkalmazás alapvető kinézetét, így egy ilyen oldalon olyan dolgok kerülnek elhelyezésre, mint fejlécek, menük, design elemek, de tartalmazhat HTML kódot, vezérlőket és tetszőleges programkódot is. Egy master page kötelezően kell, hogy tartalmazzon content placeholder-eket. Ezek határozzák meg a master page azon részeit, amelyek a weboldalak közötti navigálás során módosulnak.

A content page-ek maguk a weboldalak, vagyis a tartalom megjelenítéséért felelősek. Egyetlen content page sem állhat önmagában, mindegyikhez szükséges egy master page-t rendelni. Master page létrehozásakor az oldal kiterjesztése „master” lesz és elsőként a Visual Studio létrehozz benne a szükséges direktívát:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeBehind="Private.master.cs" Inherits="Library.MasterPages.Private" %>
```

Alkalmazásunkban négy master page-t és számos content page-t hoztunk létre. Master oldalaink: Public.master, Private.master, Employee.master, Admin.master. Mindegyikhez hozzárendeltük a CSS fájlunkat, így biztosítva az azonos megjelenést. Az egyetlen eltérés a master oldalakon elhelyezett menü vezérlőkben van, amelyek master oldalanként eltérő menüpontokat is tartalmaznak. Ezzel biztosítottuk a különféle szerepkörrel és hozzáférési móddal rendelkező felhasználók számára a jogosultságuknak megfelelő menüpontok elérését. A master oldalakhoz létrejöttükkor automatikusan hozzáadódik két contentpalceholder vezérlő. Az egyik a „head” nevű, amely belsejében lehetőségünk van metaadatok megadására, mint például a keresőknek szánt, az oldallal kapcsolatos kulcsszavak, de ide kerülnek a használni kívánt CSS fájlok elérési útvonalai is. A másik a lap törzsét reprezentálja, ide töltődnek majd be a tényleges tartalmat hordozó content oldalak. Ahhoz, hogy egy content page-hez master page-t rendeljünk, az oldal Page direktívájában meg kell adnunk a használni

kívánt master oldal elérési útvonalát. Ezt vagy abszolút útvonalként adjuk meg, vagy csak a file nevét írjuk be. Utóbbi esetben a master oldalt a rendszer az alkalmazáson belül fenntartott MasterPages almappában fogja keresni. Egy a Page direktívák közül:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPages/Public.Master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="Library.Default" Title="Könyvtár" %>
```

Fontos szabály, hogy mivel egy content page nem jeleníthető meg master page nélkül, így nem is tartalmazhat olyan szabvány HTML tageket mint <html>, <head>, <body>, mivel ezeket már definiáltuk a master-ben. Egy content oldal alapvetően content vezérlőket kell, hogy tartalmazzon, méghozzá pontosan annyit ahány contentplaceholder található a hozzá rendelt master page-ben. A content vezérlő egyik tulajdonsága a ContentPlaceHolderID, amelynek értékül kell adnunk annak a contentplaceholdernek az azonosítóját, amelyikbe szeretnénk, hogy az adott content vezérlő által tartalmazott tartalom betöltésre és megjelenítésre kerüljön:

```
<asp:Content ID="headContent" ContentPlaceHolderID="head" runat="server">
</asp:Content>

<asp:Content ID="mainContent" ContentPlaceHolderID="mainContentPlaceHolder"
runat="server">
</asp:Content>
```

5. Felhasználók kezelése

5.1 Membership

A webes alkalmazások egyik közös jellemzője a felhasználói adatok tárolásának és elérésének módja. Ez általában a felhasználói táblák létrehozásából, az adatok mozgatásához és módosításához szükséges utasítások megírásából, majd a szükséges webes felületek kialakításából áll. Az ASP.NET mindezekre egyedi és egységes megoldás nyújt. Az egyik a Membership API (Application Programming Interface), a másik pedig a profilok használata. Mivel ezen eszközök levették a vállunkról a felhasználók kezeléséhez szükséges háttér

megtervezésének és létrehozásának terhét, feladatunk az volt, hogy megtanuljuk a használatukat, majd alkalmazzuk őket saját alkalmazásunkban. Mindkét eszköz alapját automatikusan generált táblázatok és az adatkezeléshez szükséges előre megírt adatbázis utasítások, osztályok és metódusok adják.

Nézzük, először milyen funkciókkal rendelkezik a Membership API!

- használatával lehetőségünk nyílik a felhasználók létrehozására és törlésére, akár programkódból, akár az erre szolgáló konfigurációs felületen keresztül.
- jelszavakat generáltathatunk és újrageneráltathatunk, amelyekről a felhasználó automatikus üzenetet kap, amennyiben az adatbázisban található az adott felhasználóhoz rendelve e-mail cím.
- felhasználók keresése, adataik lekérdezése, módosítása.

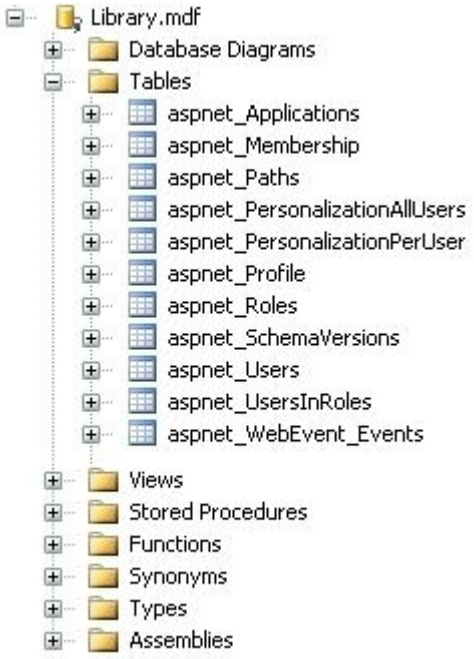
A membership használatához egy parancsot szükséges kiadnunk a megfelelő kapcsolókkal, illetve be kell konfigurálnunk a web.config fájl erre vonatkozó részét. A futtatandó fájl alapesetben az alábbi helyen található:

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_regsql.exe

Amennyiben - ahogy a mi esetünkben is - egy adatbázis fájlban szeretnénk létrehozni a membership, a profilok és a szerepkörök használatához szükséges eszközöket, az alábbi módon kell a parancsot használni:

aspnet_regsql -A all -C "Data Source=.\SQLEXPRESS;Integrated Security=True;User Instance=True" -d "C:\MyProject\APP_DATA\Library.mdf"

Ezzel létrehozásra kerülnek a szükséges dolgok a megadott .mdf adatbázis-fájlban. A parancs kiadása után a file a következő táblákat tartalmazza:



4. ábra

Adatbázistáblák a membership, profilok és szerepkörök használatához

A web.config membership szekciója:

```
<membership defaultProvider="MemberShipProvider">
      <providers>
        <add name="MemberShipProvider"
             applicationName="Library"
             requiresQuestionAndAnswer="false"
             requiresUniqueEmail="false"
             passwordFormat="Encrypted"
             connectionStringName="Library"
             enablePasswordRetrieval="true"
             enablePasswordReset="true"
             minRequiredPasswordLength="6"
             minRequiredNonalphanumericCharacters="0"
             type="System.Web.Security.SqlMembershipProvider" />
      </providers>
</membership>
```

5.2 Profilok

A membership használata csak olyan adatok kezelését teszi lehetővé, mint a felhasználónév, jelszó, e-mail cím, így mivel nekünk a felhasználók személyes adataira is szükségünk van, ezért a membership rendszert egy másik eszközzel, a felhasználói profil használatával kellett kombinálnunk. A profilok lehetőséget nyújtanak általunk megadott, tetszőleges számú és típusú felhasználói adat kezelésére. Miután az adatbázisunk készen állt a profilok használatára, az alábbi beállításokra volt szükség a web.config-ban:

```
<profile defaultProvider="SqlProvider" enabled="true">
      <providers>
        <clear/>
        <add name="SqlProvider"
type="System.Web.Profile.SqlProfileProvider" connectionStringName="Library"
applicationName="Library"/>
      </providers>
      <properties>
        <add name="FirstName" type="String"/>
        <add name="SurName" type="String"/>
        <add name="BirthPlace" type="String"/>
        <add name="BirthDate" type="String"/>
        <add name="Address" type="String"/>
        <add name="Email" type="String"/>
        <add name="PhoneNum" type="String"/>
        <add name="IdentityNum" type="String"/>
        <add name="EndOfReg" type="String"/>
      </properties>
    </profile>
```

5.3 Regisztráció

Rendszerünkben három felhasználó típust különböztetünk meg: adminisztrátor, dolgozó és ügyfél. Az adminokról és dolgozókról a rendszer csak membership adatokat tárol, míg az ügyfelek esetében mind membership, mind pedig az előbb ismertetett profilban meghatározott személyes adatok is rögzítésre kerülnek. Az admin jogosultságú felhasználók belső felületre történő belépéshez szükséges adatai már a fejlesztés során elhelyezésre kerültek a rendszerben. Dolgozót csak admin jogosultságú személy regisztrálhat a „Dolgozók” menüpont alatt található űrlap segítségével. Ugyanitt lehet megtekinteni az

összes, már beregisztrált dolgozót, és lehetőség van a törlésükre, illetve adataik módosítására. Az ügyfelek regisztrációja a dolgozók feladata és joga, a számukra létrehozott belső oldalon található „Regisztráció” menüpont segítségével. A regisztrációs űrlapon az e-mail címen kívül, minden mezőt kötelező kitölteni. Validator vezérlők ügyelnek arra, hogy ne maradhasson üresen egyetlen szükséges mező sem. Ezek olyan speciális vezérlők, amelyek egyéb vezérlőkhöz – ez esetben textbox-okhoz – vannak rendelve, figyelmeztetnek, és nem engedik tovább a munkamenetet, ha az általuk figyelt helyen nem megfelelő állapot található. Az általunk is leggyakrabban használt típusa a RequiredFieldValidator, amely nem engedi meg, hogy üres maradjon az általa figyelt mező. A RegularExpressionValidator - mint neve is mutatja -reguláris kifejezéseket vesz alapul az ellenőrzéshez. Az e-mail cím megadására szolgáló textbox-ot egy ilyen vezérlő figyeli, ügyelve arra, hogy megfelelő formátuban kerüljön a textbox-ba az e-mail cím. Esetünkben ez így néz ki:

```
<asp:RegularExpressionValidator ID="emailRegExValidator" runat="server"
ErrorMessage="Hibás formátum!" ControlToValidate="emailTextBox"
ValidationExpression=".*@.{2,}\..{2,}"
Display="Dynamic"></asp:RegularExpressionValidator>
```

Beállításra került a hibaüzenet szövege (ErrorMessage), a regex amely a formátumot adja (ValidationExpression), a vezérlő amelyet a validator „figyel” (ControlToValidate) és a megjelenítés módja (Display), amely „Dynamic” értéket kapott, így a validator üzenete nem foglal állandóan helyet az odalon, csak akkor adódik az oldalhoz ha a validálás sikertelen, vagyis ha az e-mail cím formátuma nem megfelelő. Kis eltérésekkel, hasonló módon kell beállítani más típusú validatorokat is.

Az e-mail cím megadásáról az ügyfél dönthet, ha nem kerül megadásra, a felhasználói profil úgy is létrejön, de nem generálódik jelszó az ügyfél számára, így nem fogja tudni elérni az ügyfelek részére kialakított belső oldalakat. Az e-mail cím egyébként pótolható a dolgozók által. A személyes adatok megadásán kívül meg kell adni az új ügyfél olvasójegyének számát, amely a továbbiakban az ügyfél elsődleges azonosítója lesz. Ezen kívül ki kell választani, hogy mennyi időre kíván beiratkozni. Ez három, hat, illetve tizenkét hónap lehet.

Az adatok megadása után a „Regisztráció” gombra kattintva kezdődik meg a háttérben a regisztrációs folyamat. Működésbe lépnek a validatorok és ha minden szükséges mező

kitöltésre került, megkezdődik az űrlapon szerpelő adatok begyűjtése. A folyamat egy ellenőrzéssel folytatódik:

```
ProfileInfoCollection profile =
ProfileManager.FindProfilesByUserName(ProfileAuthenticationOption.All,
cardnumTextBox.Text);
```

Ennek nyomán a ProfileInfoCollection típusú profile objektumba kerülnek mindazok az ügyfelek, akik azonos olvasójegyszámmal rendelkeznek, mint az éppen regisztrálni kívánt ügyfél. Természetesen nem lehet két ügyfélnek azonos olvasójegyszáma, így a folytatásban egy „`if (profile.Count == 0)`” feltétel vizsgálja, hogy a kollekció üres-e. Ha nem üres egy hibaüzenet jelenik meg az oldalon, miszerint a felhasználónév már foglalt. Ha üres, akkor létrejön az új ügyfél-profil, a következő módon:

```
ProfileBase newUser = ProfileBase.Create(cardnumTextBox.Text, true);
```

Ezután beállításra kerülnek a profil általunk megadott tulajdonságai, a SetPropertyValue metódus segítségével, amely a property nevét és az ahhoz tartozó értéket várja paraméterként. Példa a felhasználónév beállítására:

```
newUser.SetPropertyValue("FirstName", firstnameTextBox.Text.Trim());
```

Szükség van még a membership beállítására is. Ehhez előbb egy jelszót generáltatunk, majd az előbb használt felhasználónévvel létrehozzuk a felhasználót ott is:

```
string password = Membership.GeneratePassword(10, 0);

Membership.CreateUser(newUser.UserName, password);
```

Ezt követi a a felhasználó hozzáadása a „Private” szerepkörhöz:

```
Roles.AddUserToRole(newUser.UserName, "Private");
```

Ha meg lett adva e-mail cím, akkor egy üdvözlő üzenet jön létre és kerül elküldésre a megadott címre, benne a belépéshez szükséges jelszóval. Ennek módját az „Üzenetkezelés” című fejezetben ismertetjük.

Utolsó lépésként elmentésre kerül a profil, a Save metódus hívásával, ezzel a regisztrációs folyamat lezárult, annak sikerességéről vagy sikertelenségéről az oldal alján megjelenő üzenet tájékoztatja a felhasználót.

5.4 Belső oldalak elérése

A belső oldalakat a beléptető oldalon keresztül lehet elérni, amelyet „Belépés" menüpont alatt találunk. Itt az ASP.NET login vezérlőjét használtuk, amelyet testre szabtunk a saját igényeinknek megfelelően. A belépéshez ügyfeleknek az olvasójegyük számát és jelszavukat, dolgozóknak és adminisztrátoroknak a felhaszálónevüket és jelszavukat kell megadniuk. A login vezérlő gondoskodik a belépés során felmerülő hibák esetén a hibaüzenetek kiírásáért, amelyeket mi állítottunk be. Bejelentkezéskor a rendszer megvizsgálja, hogy létezik-e a megadott felhasználónév, majd megnézi egyezik-e az ahhoz tartozó jelszó, megtörténik a felhasználó autentikálása. A szerepköröket mi kifejezetten a felhasználók csoportosítására használtuk.

Miután az autentikáció megtörtént, megvizsgáltatjuk, hogy a belépni szándékozó felhasználó melyik szerepkörbe tartozik. Az alábbi metódus az autentikáció megtörténte után a tényleges beléptetés vagyis a user belső oldalra történő átirányítása előtt fog lefutni:

```
protected void Login1_LoggingIn(object sender, LoginCancelEventArgs e)
{
      switch (Roles.GetRolesForUser(Login1.UserName)[0])
      {
            case "Admin": Login1.DestinationPageUrl =
                              "~/ContentPages/Admin/ANews.aspx"; break;
            case "Employee": Login1.DestinationPageUrl =
                              "~/ContentPages/Employee/ENews.aspx"; break;
            case "Private": Login1.DestinationPageUrl =
                              "~/ContentPages/Private/PNews.aspx"; break;
            case "Inactive": e.Cancel = true;  break;
      }
}
```

A szerepkörétől függően irányítódik át a felhasználó a megfelelő belső oldalra. Ez azért fontos, mert így meghatározódik az, hogy ki-ki az alkalmazás melyik részéhez fog tudni

hozzáférni, vagyis megtörténik az autorizáció. Amennyiben valaki az inaktív szerepkörbe tartozik, úgy regisztrációja lejárt és nem érheti el profilját, így az 'e' argumentum „Cancel" propertyjének segítségével állítjuk le a beléptetés folyamatát.

5.5 Ügyfelek elérése

A rendszerben regisztrált ügyfelek adatait mind a dolgozók, mind pedig az adminisztrátorok el tudják érni. Az erre a célra kialakított felületet az Adatbázisok-Ügyfelek menüpont alatt található. Itt lehetőség van olvasójegy szám és vezetéknév szerint keresni. A találatok egy gridview-ba kerülnek, ahol egyenként lehet kiválasztani vagy törölni az ügyfeleket, de látható egy „Üzenet" gomb is, amelyre kattintva a felhasználó átirányítódik az üzenetküldő felületre. Az ügyfelek kereséséhez és megjelenítéséhez egy eddigiektől eltérő technikát alkalmaztunk. Ez alapvetően három eszköz létrehozásából áll. Szükséges egy úgynevezett „Data Package" megírása, ami nem más, mint egy olyan osztály, amelynek adattagjai a feladathoz szükséges adatokat jelképezik, konstruktora pedig ezen adatok értékeinek a beállítására szolgál. Esetünkben ezek az ügyfelekről tárolt személyes adatok. Szükség van még tárolt adatbázis-utasítások megírására, amik segítségével az adatok mozgatása történik. Mivel mi a beépített profilokkal dolgoztunk, így erre nem volt szükségünk, az összes adatbázissal kapcsolatos utasítást a rendszer a profilok beállításakor elkészítette és letárolta. Harmadikként pedig szükség van egy „Data Utility" osztályra, amiben a felhasználók adatain végzendő műveleteket reprezentáló metódusok kerülnek megírásra. Mivel nekünk a három komponens közül kettőnek a megírására volt szükségünk, úgy döntöttünk, ezt egyetlen osztályban írjuk meg, ez lett a Customer osztály. Osztályunk adattagjai az ügyfelek személyes adatai, konstruktora ezen adatok beállítására szolgál, metódusai pedig a következők:

- `public static Customer GetUser(string userName)`

  Statikus metódus, vagyis az osztály példányosítása nélkül is használható. Egy felhasználónevet paraméterként megadva visszatér az adott felhasználó adatait tartalmazó customer objektummal.

- `public Customer GetActUser(string userName)`

  Az aktuálisan bejelentkezett ügyfél adatait tartalmazó objektummal tér vissza. Használata az ügyfelek saját profiljainál szükséges.

- `public List<Customer> GetUsersByName(string surName)`

  Az ügyfelek vezetéknevük szerinti keresését végző metódus. Az adott vezetéknevű ügyfeleket tartalmazó listát ad vissza.

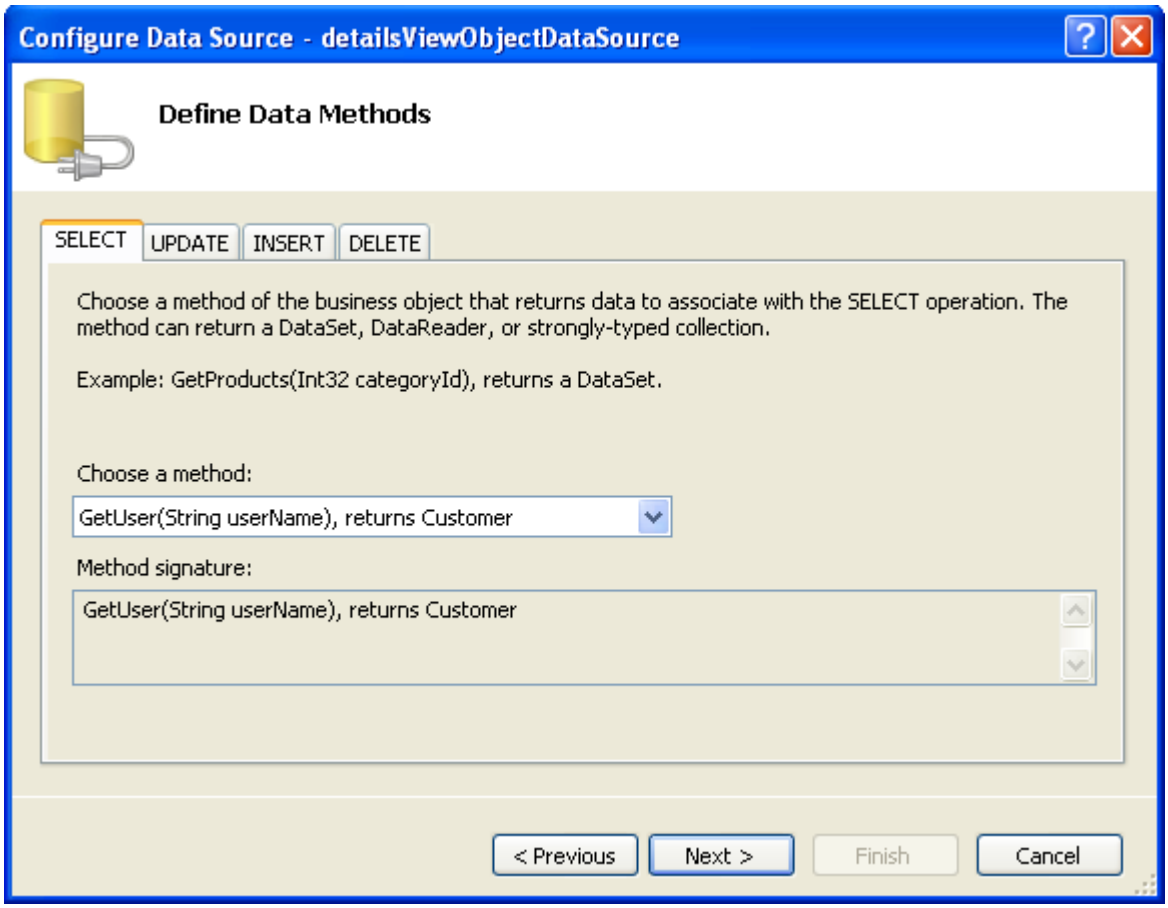
- `public void UpdateUserPrivate(Customer C)`

  Az ügyfelek saját profiljukban történő módosítását végzi.

- `public void UpdateUserEmployee(Customer C)`

  Az ügyfelek adatainak dolgozók általi módosítására szolgál.

Az adatok ideiglenes forrásául szolgáló Customer osztály és a megjelenítést végző vezérlő között a kapcsolatot egy ObjectDataSource vezérlő látja el. Ennek a vezérlőnek a működése hasonló a korábban ismertetett sqldatasource-hoz, azzal a különbséggel, hogy az adatokat nem közvetlenül az adatbázisból olvassa ki, hanem egy tetszőleges objektumból. Ehhez az objektum metódusait használja, jelen esetben ezek a Customer osztály előbb ismertetett metódusai. Az adatbázissal való kommunikáció és az adatok kezelése abban az osztályban kerülnek megírásra, amelynek példánya az objektum, így tehát az objectdatasource nem áll közvetlen kapcsolatban az adatbázissal.



5. ábra

ObjectDataSource lekérdezésének beállítása.

A megjelenítendő rekord elemei a GetUser metódus által visszaadott példány adattagjai.

A gridview-ban megjelenő ügyfelek közül egyet kiválasztva az oldalon egy detailsview vezérlő jelenik meg, benne a kiválasztott ügyfél adataival. A „Módosításra” kattintva a detailsview módosítható állapotba kerül, ezzel a mezők szerkeszthetővé válnak. A dolgozóknak az összes ügyféladat módosítására jogosultságuk van. Itt van lehetőség a regisztráció meghosszabbítására is. Az adminisztrátor számára hasonló felület áll rendelkezésre, azzal a különbséggel, hogy az adminisztrátor jogosult az ügyfelek adatbázisból való törlésére.

5.6 Saját profil

Ügyfélként belépve a privát szerepkörben elérhető belső oldalra kerül a felhasználó. Itt a „Profil” menüpont alatt érheti el a saját adatait tartalmazó táblázatot. Az adatokat ebben az esetben is detailsview jeleníti meg. Az adatbázisból az ügyfél adatait tartalmazó rekord a Customer osztály egy példányába kerül. A műveletet egy objectdatasource végzi, és biztosítja a kapcsolatot az adatokat megjelenítő detilsview-val is. Az aktuálisan bejelentkezett felhasználó profiljának lekérdezése az alábbi módon történik:

```
ProfileBase actProfile = HttpContext.Current.Profile;
```

Ebből a profilból a GetActUser metódus készít egy Customer típusú objektumot, majd visszatér azzal, így az átkerül az objectdatasourcehoz, ami pedig a detailsview adatforrása, így az feltöltődik a személyes adatokkal.

Az ügyfelek korlátozottan módosíthatják adataikat. Jelenlegi állapot szerint csak e-mail címüket és telefonszámukat módosíthatják saját maguk, egyéb adatok változtatását csak a dolgozók és adminisztrátorok végezhetik. Az ügyfél itt tájékozódhat a regisztrációjának végét jelző dátumról is.

A fejlesztés során a felhasználók kezelésének kialakítása tűnt az egyik legbonyolultabb és legösszetettebb feladatnak. Próbáltunk egyre jobban elmélyedni a membership, a profilok és szerepkörök nyújtotta lehetőségekben, kezdetben természetesen csak elméleti szinten. Amikor a tényleges megvalósításra került a sor, akkor szembesültünk egy, az ASP.NET legújabb verziójában fellelhető hibával. Ehhez tudni kell, hogy az ASP.NET megkülönböztet „Web Site” és „Web Application” projektet. A különbség a projekt méretében, szerkezetében és részben működésében van. Utóbbi ajánlott nagyobb alkalmazások készítéséhez, mi is ennek a használatát választottuk. A profilok konfigurációját követően a Visual Studio automatikusan létrehoz egy erősen típusos „Profile” objektumot, amely minden, az alkalmazás által tartalmazott oldal kódjában elérhetővé válik. Ezen objektum property-jeit adják azután azok a property-k, amelyeket a web.config-ban a profil beállításánál megadtunk. Így egyszerűen lekérdezhetők és beállíthatók egy felhasználó

profiljának adatai. A hiba abban rejlik, hogy a „Profile” objektum csak „Web Site” esetén jön létre, így a mi esetünkben nem volt elérhető. Találtunk ugyan egy olyan külső forrásból letölthető eszközt, ami majdnem teljes egészében helyettesíti a „Profile” objektumot, de mi szerettünk volna a meglévő eszközökkel megoldani a feladatot, így ennek használatától eltekintettünk. Ily módon kicsit körülményesebb módon, kis „kerülővel” tudtuk elérni és kezelni a felhasználói profilokat, ami nagyobb odafigyelést igényelt.

Az eszköz elérhető az alábbi címen: http://code.msdn.microsoft.com/WebProfileBuilder

6. Könyvek kezelése

6.1 Keresés a katalógusban

A katalógus mögött a Books, Loan, Reserve táblák állnak. Ezek rendre a könyvekért, kölcsönzésekért és foglalásokért felelősek. A keresés mind a három szerepkörben különbözik. Publikus szinten csak a könyvek táblájának lekérdezésére van lehetőség. Privát szerepkörben DataGridView-ba beépített vezérlők felelősek azért, hogy a felhasználók lefoglalhassanak könyveket.

Továbbá lehetőségük van a könyvek részletes adatainak megtekintésére egy DetailsView segítségével. A Kölcsönzések oldalon figyelemmel követhetik a felhasználók saját kölcsönzéseiket, foglalásaikat, utóbbiakat itt vissza is vonhatják.

A könyvtáros szerepkörben lehetőség adódik az alapvető DML (Data Manipulation Language) utasításokra, mint például SELECT, UPDATE, INSERT és DELETE.

A katalógusban a felhasználóknak összetett keresésre van lehetőségük. A kereső oldalon található egy DropDownList, TextBox, CheckBox és mégegy DropDownList. Az első vezérlő felelős azért, hogy kiválasszuk, hogy a könyv mely attribútumára akarunk rákeresni. A TextBox-ba írhatjuk a keresendő karaktersorozatot. A következő vezérlő (CheckBox) segítségével megadhatjuk, hogy a keresés során a keresendő karaktersorozatot csak tartalmazza az aktuális attribútum értéke vagy pontos egyezés esetén legyen eredmény.

A vezérlők ily módon négy sorban helyezkednek el, melyet az utolsó DropDownList-tel lehet logikai és-vagy operátorokkal összekötni. A négy sor közül legalább egyet kötelező kitölteni a keresés sikeressége érdekében, a többi opcionális. Ennek ellenőrzését a CustomValidator végzi. CustomValidator–hoz kötött metódus nem csak a bemenet helyességéért felelős, ez végzi a beállított vezérlők szerinti Transact-SQL parancsok generálását is, mely egy StringBuilder típusú sBuilder mező állapotát fogja meghatározni. Ezt a mezőt az SqlDataSource tulajdonságainak adjuk értékül, mely connectionstring segítségével kapcsolódik az adatbázishoz és visszatér a tranzakció eredményével. Az eredményt egy DataGridView jeleníti meg, melynek DataSource tulajdonsága (property) fogja megkapni az SqlDataSource-t értékül. Majd a DataGridView DataBind metódusa köti az adatokat a DataGridView-hoz. A keresés megkezdése előtt lehetőségünk van a DataGridView-ban megjelenő rekordok mennyiségének beállítására. A DataGridView-ban lapozás lehetséges, ha esetleg túl sok az adat vagy kevés értéket állítottunk be az előbbi opciónál. A vezérlő HeaderText-jére kattintva pedig attribútum szerint rendezhetjük a megjelent tételeket.

Publikus hozzáférés esetén a felhasználóknak lehetősége van bejelentkezés nélkül megnézni a könyvtár katalógusát. A kód ellenőrzi, majd az igényeknek megfelelően legenerálja a lekérdezést.

A validator_ServerValidate metódus aktuális paraméterlistájában található args paraméter IsValid tulajdonsága kapja értékül a valid módszer értékét. Ez az érték fogja meghatározni, hogy a vezérlőkhöz kötött minimális követelmény teljesül-e.

```
        protected void validator_ServerValidate(object source,
ServerValidateEventArgs args)
        {
            args.IsValid = valid();
        }
```

A numberOfTextBox metódus felelős azért, hogy keresés során ellenőrizze, a felhasználó hány darab textBox-ot töltött ki, majd ennek az értékével tér vissza. Ha a módszer visszatérési értéke 0, akkor az args.IsValid tulajdonság is false értéket fog kapni, így a keresés nem következik be.

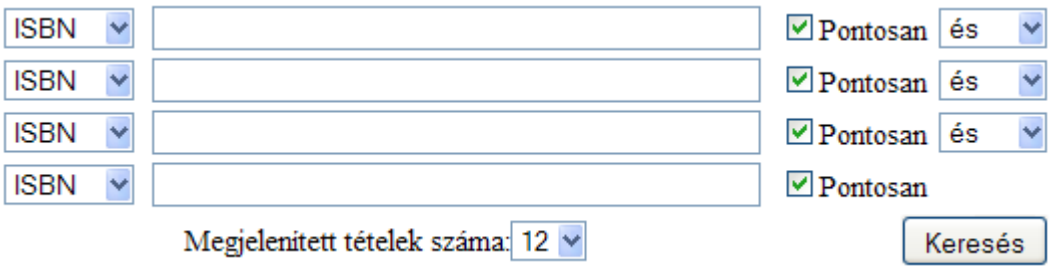
```
private int numberOfTextBox()
{
    int a = 0;
    if (catTextBox1.Text != "")
        ++a;
    if (catTextBox2.Text != "")
        ++a;
    if (catTextBox3.Text != "")
        ++a;
    if (catTextBox4.Text != "")
        ++a;
    return a;
}
```

A valid metódus végzi magát a lekérdezés generálását és ellenőrzi beállítások helyességét. Visszatérési értéke false, ha numberOfTextBox visszatérési értéke 0, egyébként true. Ha legalább egy textBox ki van töltve, akkor ellenőrzi ezt, majd a textBox-szal egy sorba tartozó vezérlők értékei szerint generálja az utasítást. Ha több textBox Text property-je sem üres, akkor az utolsó vezérlő szerint OR vagy AND operátorral kapcsolja össze.

```
private bool valid()
{
    int a = numberOfTextBox();
    if (a == 0) return false;
    if (catTextBox1.Text != "")
    {
        sBuilder.Append(catDropDownList1.SelectedValue);
        if (!catCheckBox1.Checked)
            sBuilder.Append(" LIKE '%" + catTextBox1.Text + "%' ");
        else
            sBuilder.Append(" = '" + catTextBox1.Text + "' ");
        if (a > 1)
        {
            sBuilder.Append(andOrList1.SelectedValue);
            --a;
        }
    }
    if (catTextBox2.Text != "")
    {
        sBuilder.Append(catDropDownList2.SelectedValue);
        if (!catCheckBox2.Checked)
            sBuilder.Append(" LIKE '%" + catTextBox2.Text + "%' ");
        else
            sBuilder.Append(" = '" + catTextBox2.Text + "' ");
        if (a > 1)
        {
            sBuilder.Append(andOrList2.SelectedValue);
            --a;
        }
    }
    if (catTextBox3.Text != "")
    {
        sBuilder.Append(catDropDownList3.SelectedValue);
```

```
            if (!catCheckBox3.Checked)
                sBuilder.Append(" LIKE '%" + catTextBox3.Text + "%' ");
            else
                sBuilder.Append(" = '" + catTextBox3.Text + "' ");
            if (a > 1)
            {
                sBuilder.Append(andOrList3.SelectedValue);
                --a;
            }
        }


        if (catTextBox4.Text != "")
        {
            sBuilder.Append(catDropDownList4.SelectedValue);
            if (!catCheckBox4.Checked)
                sBuilder.Append(" LIKE '%" + catTextBox4.Text + "%' ");
            else
                sBuilder.Append(" = '" + catTextBox4.Text + "' ");
        }
        return true;
    }
```



6. ábra

A keresőfelület.

A keresőfelület minden szerepkörben megegyezik, a különbség a DataGridView felépítésében van. Privát részen a DataGridView három gombbal bővül. Az első a kiválasztás, mely az aktuális rekord DetialsView-ban való megjelenítését teszi lehetővé. A DetialsView-ban a kiválasztott könyvről jelenik meg részletes információ. A következő két gomb a kölcsönzés és a lefoglalás, mely nevükből adódóan az aktuális rekord kölcsönzését és foglalását teszi lehetővé. A dolgozó szerepkörben a DataGridView kiválasztás, módosítás és törlés gombbal bővül. A kiválasztás során minden részletes információ megjelenik az adott rekordról. Módosítás estén lehetőségünk van a rekordhoz tartozó attribútumok értékeinek

megváltoztatására, míg törlés esetén törli az adott könyvet a táblából. A törlés feltétele, hogy a könyv státusza elérhető legyen. Itt szintén SqlDataSource-t használtunk az adatbázishoz való kapcsolódáshoz, illetve az adatok DataGridView-hoz való kötéséhez. A kereséshez hasonlóan az utasításokat a SqlDataSource tulajdonságaihoz kötjük, melyek string típusú értékeket várnak. Értelemszerűen a SelectCommand-hoz a SELECT, UpdateCommand-hoz az UPDATE, illetve DeleteCommandhoz a DELETE utasítást kötjük. Innentől az SqlDataSource feladata az utasítások végrehajtása.

7. Kölcsönzések kezelése

A kölcsönzések kezelésére csak a dolgozóknak van lehetősége és jogosultsága. Erre szolgál a Kölcsönzések menüpont. Az itt található oldalon az összes aktuális kölcsönzés listázásra kerül, amely rekordok között keresni lehet a könyv azonosítója, vagy az ügyfél olvasójegyének száma alapján. Található itt egy kölcsönző űrlap, amelynek a kikölcsönözni kívánt könyv azonosítóját és a kölcsönző ügyfél olvasójegyének számát kell megadni. Kölcsönzéskor a Books tábla State attribútuma módosul kölcsönzöttre, illetve bekerül a Loan táblába az új kölcsönzés. Amikor a könyvet visszahozzák, a kölcsönzéseket listázó gridview-ban kell az adott kölcsönzést megkeresni, majd a „Vissza” gombra nyomni. Könyv foglalását az ügyfél saját belső felületén kezdeményezheti. Ekkor a State attribútum foglalt állapotba kerül, illetve a Reserve táblába bekerül az új rekord. Kölcsönzési és foglalási történelmét a Kölcsönzések menüpont alatt tudja követni a felhasználó, ahol foglalásait visszavonhatja.

8. Üzenetkezelés

8.1 Hírek

Rendszerünkben a dolgozók és az adminisztrátorok postázhatnak új híreket, amelyek a főoldalon, illetve a belső oldalakon lévő menükből is elérhetőek.

Egy dolgozó a Hírek-Új hír feladása menüpont alatt elérhető űrlapon keresztül adhat fel új hírt. Ugyanitt jelennek meg az aktuális dolgozó által eddig feladott hírek, amelyek közül bármelyik törölhető. Egy hírt kiválasztva a hír adatai és szövege az alatta található űrlap mezőibe töltődik, így az elolvasható és tetszés szerint módosítható. Egy hír feladásához a hír címét az érvényesség kezdetét jelző dátumot és a hír szövegét kell megadni. Az érvényesség végét jelző dátum megadása nem kötelező, amennyiben nem kerül beírásra, úgy a hír korlátlan ideig érvényes lesz. Az adminisztrátor a „Hírek" menüpont alatt az összes feladott hírt láthatja, amelyek közül bármelyiket törölheti és módosíthatja. Itt található még egy nyomógomb, amely segítségével a lejárt hírek törölhetők az adatbázisból.

8.2 Üzenetek

Jelenlegi állás szerint az üzenetkezelés rendszerünkben egyirányú. Ez azt jelenti, hogy a dolgozók küldhetnek üzenetet az ügyfeleknek a belső üzenetküldő felületről, amelyet aztán az ügyfél e-mail formájában fog megkapni. Természetesen ez csak akkor lehetséges, ha az ügyfél regisztrációkor megadta e-mail címét. Ha ez nem történt meg, akkor erről egy hibaüzenet tájékoztatja a dolgozót, amikor az üzenetet el kívánja küldeni. Az „Üzenetkezelés" menüpont alatt tekintheti meg a dolgozó az általa elküldött üzeneteket és lehetősége van azok törlésére is. Üzenet küldéséhez az űrlapon meg kell adni a címzett olvasójegy számát és az üzenet szövegét, a tárgy mező kitöltése opcionális. A küldés sikerességéről az oldal alján megjelenő üzenet tájékoztatja a felhasználót. Sikeres küldés esetén az üzenet adatai a küldő személy azonosítójával együtt az adatbázis „Messages" táblájába kerülnek, és innen történik az elküldött üzenetek lekérdezése is, az aktuálisan bejelentkezett dolgozó azonosítója alapján.

A műveleteket egy sqldatasource vezérlő végzi. Minden üzenet, mint rendszerüzenet egyetlen e-mail címről kerül elküldésre. Ezt a címet a Gmail rendszerben regisztráltuk, konyvtar.szakdolgozat@gmail.com címen. Mind az automatikus rendszerüzenetek, mind pedig a dolgozók által küldött üzenetek küldését az általunk írt „Email” osztály végzi. A programkódból történő e-mail küldéséhez a .NET SmtpClient osztályát, ennek Credentials tulajdonságának beállításához a NetworkCredential osztályt, valamint a MailMessage osztályt használtuk.

Email osztályunknak két konstruktora és egy túlterhelt Send nevű metódusa van, amely a tényleges küldést végzi. Egyik konstruktora paraméterezett, másik paraméter nélküli. Előbbinek már példányosításkor megadhatjuk a címzett e-mail címét, az üzenet tárgyát és tartalmát, amely adatok segítségével a konstruktor beállítja a küldéshez szükséges információkat. Az osztály adattagjai és konstruktorai:

```
        private string from, to, subject, content, host;
        private NetworkCredential authentication;
        private SmtpClient mailClient;

        public Email(string to, string subject, string content)
        {
            host = "smtp.gmail.com";
            from = "konyvtar.szakdolgozat@gmail.com";
            this.to = to;
            this.subject = subject;
            this.content = content;

            authentication = new
NetworkCredential("konyvtar.szakdolgozat@gmail.com", "diffiehellman");

            mailClient = new SmtpClient(host, 587);
            mailClient.EnableSsl = true;
            mailClient.UseDefaultCredentials = false;
            mailClient.Credentials = authentication;
        }

        public Email()
        {
            host = "smtp.gmail.com";
            from = "konyvtar.szakdolgozat@gmail.com";

            authentication = new
NetworkCredential("konyvtar.szakdolgozat@gmail.com", "diffiehellman");

            mailClient = new SmtpClient(host, 587);
            mailClient.EnableSsl = true;
            mailClient.UseDefaultCredentials = false;
            mailClient.Credentials = authentication;
        }
```

A két Send metódus közül az egyik szintén rendelkezik paraméterekkel, míg a másik nem. A paraméterezett konstruktor használatakor a paraméter nélküli Send metódus használandó. Ez szolgál a dolgozók által küldött üzenetek továbbítására. A másik Send metódus a rendszerüzenetek küldését végzi, mivel paraméterlistáján egy „mode” paraméterben kell megadni a küldendő rendszerüzenet típusát, amely lehet például a sikeres regisztrációt követően kiküldött automatikus üzenet, vagy az elfelejtett jelszót kiküldő üzenet.

Előbbi esetben a mode paraméter értéke „CustomerReg”, utóbbiban „PasswordRecovery” kell, hogy legyen. Mindkét metódus bool típusú értékkel tér vissza, amely értéke a küldés sikerességétől függ.

A paraméter nélküli Send metódus:

```
public bool Send()
{
    MailMessage mail = new MailMessage(from, to, subject, content);
    mail.IsBodyHtml = false;

    try
    {
        mailClient.Send(mail);

        return true;
    }
    catch
    {
        return false;
    }
}
```

9. Biztonság

A biztonságos alkalmazás kialakítására való törekvés az egyik legfontosabb szempont kell, hogy legyen már a fejlesztés megkezdésekor is. Webes alkalmazások esetén ez hatványozottan igaz, ezek ugyanis működésükből adódóan igen komoly támadásoknak lehetnek kitéve. Legfontosabb, hogy megakadályozzuk a jogosulatlan hozzáféréseket és biztosítsuk az adatok sértetlenségét. Az ASP.NET biztonságért felelős keretrendszere

hatékony eszközöket nyújt a felhasználók autentikálására és autorizációjára, így biztosítva, hogy személyek csak tényleges jogok birtokában használhassák alkalmazásunkat.

Az általunk feldolgozott szakirodalom a webes alkalmazások biztonságának négy szintjét különbözteti meg:

1. Autentikáció: ez a szint szolgál a felhasználók azonosítására, továbblépéshez a felhasználóknak különféle információkkal kell igazolniuk, hogy jogosultak a rendszer használatára.
2. Autorizáció: ezen a szinten már ismert, hogy ki az, aki a rendszert használni szeretné, a teendő annak eldöntése, hogy az illető a rendszer mely részeihez, erőforrásaihoz férhet hozzá.
3. Bizalom szintje: biztosítani kell, hogy a rendszerben dolgozó ügyfél által végzett munkafolyamatokat és az ezekhez tartozó adatokat senki ne láthassa, aki arra nem jogosult. Ezt különféle titkosításokkal kell megvalósítani. Szükség lehet a webszerver és a kliens közötti csatorna titkosítására vagy például az adatok titkosított formában való tárolására.
4. Sérthetetlenség: végül biztosítani kell, hogy a kliens és szerver között mozgó adatokat senki se módosíthassa. Erre szolgálnak például a digitális aláírások.

A fejlesztés során az első három szint megvalósításának módszereit tanulmányoztuk és alkalmaztuk projektünkben.

9.1 Forms Autentikáció

Az ASP.NET egyik autentikációs rendszere a forms autentikáció. Ez a rendszer „ticket” vagy más néven „token” alapú. Ez azt jelenti, hogy amikor egy felhasználó bejelentkezik, egy ticketet kap, ami tartalmazza az adott felhasználóra vonatkozó alapvető információkat. Ezek az információk titkosított formában, cookie-ként kerülnek csatolásra minden, a szerver felé irányuló kérelemhez. Amikor a felhasználó egy olyan oldalra lép, ahol nem engedélyezettek az anonim felhasználók, a rendszer ellenőrzi a ticket meglétét. Amennyiben nem talál ilyet, a felhasználó automatikusan a beléptető oldalra irányítódik, ha viszont talál, akkor a felhasználó megkapja a jogosultságot a lap megtekintéséhez. Ennél a típusú autentikációnál a mi feladatunk eldönteni, hogy hol szeretnénk tárolni az autentikációhoz szükséges felhasználói információkat. Legcélszerűbb ezt a web.config fájlban megtenni:

```
<authentication mode="Forms">
      <forms name="LibraryCookie"
             loginUrl="/ContentPages/Public/Login.aspx"
             timeout="20"
             slidingExpiration="true"
             protection="All">
      </forms>
</authentication>
```

Az autentikációs beállításoknál megadjuk a cookie nevét, a beléptető oldal elérési útvonalát, a cookie érvényességi idejét percben, azt, hogy az érvényességi idő minden lekéréskor újra induljon-e és a cookie titkosítási szintjét. Ezen kívül a „credentials” szekcióban kerülhetnek megadásra az egész alkalmazásra vonatkozó, állandó érvényű felhasználónevek és jelszavak.

9.2 Autorizáció

Mint ahogy a korábbi fejezetekben tárgyaltuk, a felhasználók alapvető felosztását a szerepkörök segítségével végeztük. Ezek határozzák meg azt, hogy egy adott személy az alkalmazáson belül mely részeket érheti el. Ezen kívül szükségünk van annak a beállítására, hogy kívülről, például egy böngészőben URL-ként megadva, közvetlenül ne érhessenek el belső oldalakat. Ezt is a web.config-ban történő konfigurációval érhetjük el. Az „authorization" részben lehetőségünk van beállítani az általunk tiltott vagy éppen engedélyezett felhasználók körét.

Az alábbi példa az adminisztrátorok számára kialakított belső felület eléréséhez szükséges beállításokat tartalmazza:

```
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
      <authorization>
        <allow roles="Admin"/>
        <deny users="*"/>
      </authorization>
    </system.web>
</configuration>
```

Az ezt tartalmazó web.config fájl az Admin almappában található, így hatását az ugyanitt lévő oldalakra fejti ki. A hozzáférés szabályozása az „allow" és „deny" kulcsszavakkal történik. Látható, hogy a hozzáférés engedélyezése szerepkör szerinti, azaz az Admin szerepkörbe tartozó felhasználók fogják tudni elérni az oldalakat, míg minden egyéb felhasználó számára az oldalak elérése tiltott.

10. Összefoglalás

Úgy érezzük a fejlesztés során kellően megalapoztuk mind az ASP.NET-tel, mind pedig a szoftverfejlesztéssel, csoportos munkával kapcsolatos ismereteinket. Elmondhatjuk, hogy megtanultuk az alapokat, nemcsak elméletben, hanem gyakorlati szinten is. Az ASP.NET használatával egy olyan technológiát ismerhettünk meg, ami csupa új dolgot nyújtott számunkra. Volt már részünk kisebb webfejlesztésben korábbi tanulmányaink során, ahol HTML-t, CSS-t és PHP-t használtunk, de összehasonlítva mostani munkánkkal elmondhatjuk, hogy igen jó véleménnyel vagyunk az ASP.NET-tel történő fejlesztésről. Mivel teljesen új szemlélet szükséges ehhez a technológiához, ezért a tanulási fázis hosszú lehet. Mi egy olyan átfogó elméleti alapozással kezdtünk, amelynek legfontosabb elemei az ASP.NET működésével kapcsolatos tudnivalók voltak, tehát megpróbáltuk megérteni azokat a folyamatokat, amelyek a működés során a háttérben zajlanak és egy felhasználó számára láthatatlanok maradnak. Úgy gondoljuk ez minden olyan ember számára megkerülhetetlen fázis, aki a későbbiekben használni szeretné ezt a technológiát. Ezután következtek a tényleges eszközök, a vezérlők működésének és a különféle programozási technikáknak az megismerése. A gyakorlati alkalmazás elsajátítása gyakran hosszú időbe telt, mivel tényleges kipróbálásukkor ütköztünk általában olyan problémákba, amelyek nélkülözhetetlenek voltak a működés biztos megértéséhez. Ezekben az esetekben ugyanis, gyakran hosszú próbálgatási, hibakeresési fázis következett, ami eredményeképpen fény derült rengeteg olyan hibára, amelyek felmerülhetnek az adott eszköz használata során. A hibák kiküszöbölése, a megoldás megtalálása elengedhetetlen része volt a tanulási folyamatnak. Kifejezetten előnyünkre vált az is, hogy lehetőségünk volt a csoportos munkavégzésre. Igaz ezt a munkát mindössze ketten csináltuk, sokszor volt szükség egyeztetésekre, kompromisszumokra annak érdekében, hogy egyszerre tudjunk haladni és nagyjából azonos mennyiségű munkát végezzünk. Úgy érezzük az e téren szerzett tapasztalatnak nagy hasznát tudjuk venni a későbbiekben, főleg egy munkahelyen vagy bármilyen team munka során. Elengedhetetlen része volt a fejlesztésnek a verziókövetés használatának megtanulása. Így utólag elmondhatjuk, hogy nagyon nagymértékben könnyíti meg a munkát, használatára az ember akaratlanul is rákényszerül és biztosan állíthatjuk, hogy ezentúl nem vágunk bele nagyobb fejlesztési munkába verziókövetés nélkül. Másik, számunkra nagy jelentőséggel bíró tapasztalat a

szoftverfejlesztés menetének megismerése volt. Mivel eddig nem volt dolgunk ekkora projekttel, nagyon fontos volt, hogy megfelelő részekre osszuk fel a fejlesztést, törekedve arra, hogy egyetlen kulcsfontosságú fázis se maradjon ki. Tapasztaltuk, ahogy munkánk nyomán épül a rendszer, egyre újabb funkciók kezdenek nemcsak működni, hanem jól működni, kiépülnek az egységek közötti kapcsolatok, szépen lassan „összeáll a kép”. Szoftverünk fejlettsége terveinknek megfelelő szinten van. Az alapvető, előre eltervezett funkciók stabilan működnek.

I. Függelék

Adatdefiníciós (DDL) utasítások az adattáblák felépítéséhez

- Könyvek tábla

```
CREATE TABLE Books
    (BookID int PRIMARY KEY NOT NULL,
     ISBN nchar(10) NOT NULL,
     Author nchar(30) NOT NULL,
     Title nchar(100) NOT NULL,
     Publisher nchar(30) NOT NULL,
     Year nchar(30) NOT NULL,
     State int NOT NULL,
     Available smalldatetime)
```

- Kölcsönzés tábla

```
CREATE TABLE Loan
    (LoanID int PRIMARY KEY NOT NULL,
     BookID int NOT NULL,
     UserID nchar(8) NOT NULL,
     [From] smalldatetime NOT NULL,
     [To] smalldatetime NOT NULL,
     BackDate smalldatetime)
```

- Foglalás tábla

```
CREATE TABLE Reserve
    (ReserveID int PRIMARY KEY NOT NULL,
    BookID int NOT NULL,
     UserID nchar(8) NOT NULL,
     [From] smalldatetime NOT NULL,
     [To] smalldatetime NOT NULL)
```

- Üzenetek tábla

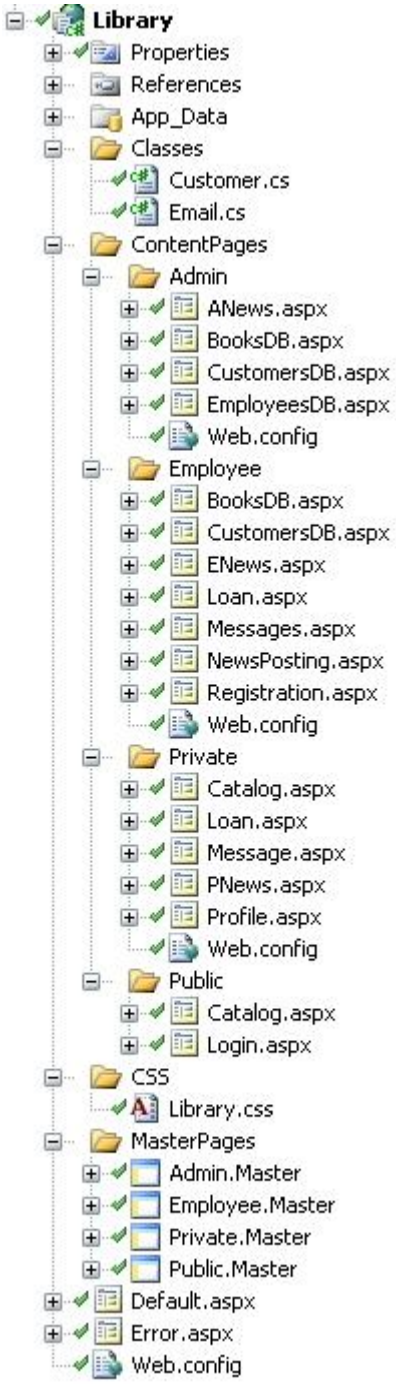
```
CREATE TABLE Messages
    (Sender nvarchar(30) NOT NULL,
     Recipient nvarchar(30) NOT NULL,
     Date smalldatetime NOT NULL,
     Subject nvarchar(50),
     Content nvarchar(MAX) NOT NULL)
```

- Hírek tábla

```
CREATE TABLE News
    ( Date smalldatetime NOT NULL,
      Title nvarchar(50) NOT NULL,
      Valid smalldatetime,
      Text nvarchar(MAX) NOT NULL,
      Sender nvarchar(30) NOT NULL)
```

II. Függelék

Az alkalmazás struktúrája



Irodalomjegyzék

Nyomtatott források:

[1] Matthew MacDonald, Mario Szpuszta: Pro ASP.NET in C# 2008. Apress, 2008.

[2] Trey Nash: Accelerated C# 2008. Apress, 2008.

Internetes források:

[3] Scott Mitchell: Security Basics and ASP.NET Support.

http://www.asp.net/learn/security/tutorial-01-cs.aspx

[4] Scott Mitchell: An Overview of Forms Authentication

http://www.asp.net/learn/security/tutorial-02-cs.aspx

[5] Scott Mitchell: Forms Authentication Configuration and Advanced Topics.

http://www.asp.net/learn/security/tutorial-03-cs.aspx

[6] Scott Mitchell: An Overview of Inserting, Updating, and Deleting Data

http://www.asp.net/learn/data-access/tutorial-16-cs.aspx

[6] Video Tutorial: Understanding Security and Network Connectivity

http://www.asp.net/learn/sql-videos/video-109.aspx

[7] Video Tutorial: Connecting your Web Application to SQL Server 2005 Express

http://www.asp.net/learn/sql-videos/video-110.aspx